**vanilla gan**

discriminatory layers

W1: 784, 128 stddev = xavier\_stddev

B1: 0s 128

W2: 128, 1 stddev = xavier\_stddev

B2: 0s 1

theta = [D\_W1, D\_W2, D\_b1, D\_b2]

generator layers

W1: 100, 128 stddev = xavier\_stddev

B1: 0s 128

W2: 128, 784 stddev = xavier\_stddev

B2: 0s 784

theta\_G = [G\_W1, G\_W2, G\_b1, G\_b2]

Noise starter:

np.random.uniform(-1., 1., size=[m, n])

generator:

relu

prob = sigmoid

return prob

discriminator:

logit = relu

prob = sigmoid

return prob and logit (probability isn’t used anywhere this code only uses the logits)

Alternative Losses:

D\_loss\_real = tf.reduce\_mean(tf.nn.sigmoid\_cross\_entropy\_with\_logits(logits=logit\_real, labels=all 1))

D\_loss\_fake = tf.reduce\_mean(tf.nn.sigmoid\_cross\_entropy\_with\_logits(logits=logit\_fake, labels=all 0))

D\_loss = D\_loss\_real + D\_loss\_fake

G\_loss = tf.reduce\_mean(tf.nn.sigmoid\_cross\_entropy\_with\_logits(logits=logit\_fake, labels=all 1))

Solvers:

D\_solver = tf.train.AdamOptimizer().minimize(D\_loss, var\_list=theta\_D)

G\_solver = tf.train.AdamOptimizer().minimize(G\_loss, var\_list=theta\_G)

**DCGAN faces**

Reset\_default\_graph

Discriminatory layers

With Tf.variable\_scope reuse

Conv2d(kernel size = 5, filters=256, strides=2, padding same, lrelu)

Dropout

Conv2d(kernel size = 5, filters=128, strides=1, padding same, lrelu)

Dropout

Conv2d(kernel size = 5, filters=64, strides=1, padding same, lrelu)

Dropout

Tf.contrib.layers.flatten

Tf.layers.dense(units-128, lrelu)

Tf.layers.dense(units=1, sigmoid)

Generator layers

With Tf.variable\_scope **don’t** reuse

Tf.layers.dense(units=4\*4\*3, lrelu)

Dropout

Batch\_norm(decay=0.9)

Reshape([-1, 4, 4, 3])

Resize([10,10])

Conv2d\_transpose(kernel size=5, filters = 256, strides = 2, padding same, lrelu)

Dropout

Batch\_norm(decay=0.9)

Conv2d\_transpose(kernel size=5, filters = 128, strides = 2, padding same, lrelu)

Dropout

Batch\_norm(decay=0.9)

Conv2d\_transpose(kernel size=5, filters = 64, strides = 1, padding same, lrelu)

Dropout

Batch\_norm(decay=0.9)

Conv2d\_transpose(kernel size=5, filters = 3, strides = 1, padding same, sigmoid)

Losses

Get results of descriminator for real and fake (fake reuse=true)

Grab the trainable variables from the generator and discriminators (theta?)

d\_reg = tf.contrib.layers.apply\_regularization(tf.contrib.layers.l2\_regularizer(1e-6), vars\_d)

g\_reg = tf.contrib.layers.apply\_regularization(tf.contrib.layers.l2\_regularizer(1e-6), vars\_g)

loss\_d\_real = binary\_cross\_entropy(all 1, result of discriminator for real)

loss\_d\_fake = binary\_cross\_entropy(all 0, result of discriminator for fakes)

loss\_d = tf.reduce\_mean(0.5 \* (loss\_d\_real + loss\_d\_fake))

loss\_g = tf.reduce\_mean(binary\_cross\_entropy(all 1s, result of discriminator for fakes))

Optimizers:

update\_ops = tf.get\_collection(tf.GraphKeys.UPDATE\_OPS)

with tf.control\_dependencies(update\_ops):

optimizer\_d = tf.train.RMSPropOptimizer(lr=0.0001).minimize(loss\_d + d\_reg, the d variables (theta?))

optimizer\_g = tf.train.RMSPropOptimizer(lr=0.0002).minimize(loss\_g + g\_reg, the g variables (theta?))

noisestarter = np.random.uniform(0.0, 1.0, [batch\_size, n\_noise]).astype(np.float32)

grab losses from session run and find mean of the d\_real\_losss and d\_fake\_loss

if the g\_loss \*1.35 is < d\_loss: train generator

if the d\_loss \*1.35 is < g\_loss: train discriminator